Aviation Electronics, Information Technology, Telecommunications, Electricals, and Controls (AVITEC)
Vol. 8, No. 1, February 2026, pp. 39-47, p-ISSN 2685-2381, e-ISSN 2715-2626
http://dx.doi.org/10.28989/avitec.v8i1.3510

Research Article

Hyperparameter Tuning of XGBoost for Flooding Attack Detection in SDN-based Vehicular Ad Hoc Networks (VANETs) under Limited Resources

Chairunisa Rahma Putri*, **Galura Muhammad Suranegara, Ichwan Nul Ichsan**
Department of Telecommunication System, Universitas Pendidikan Indonesia, Indonesia

Article Info

Article history:
Submitted October 9, 2025
Accepted November 17, 2025
Published November 29, 2025

Keywords:
Extreme Gradient Boosting (XGBoost);
flooding attack detection;
Vehicular Ad Hoc Network (VANET);
machine learning;
hyperparameter optimization;
resource constrained;
Software Defined Network (SDN).

ABSTRACT

Software-Defined Network (SDN) based Vehicular Ad Hoc Network (VANET) infrastructure network enables centralized vehicle control. However, due to its centralized nature, SDN-based VANET is vulnerable to flooding attacks such as Distributed-Denial of Services (DDoS) or Denial of Service (DoS) attacks that can disrupt network availability and endanger traffic safety. This study aims to detect flooding attacks using the Extreme Gradient Boosting (XGBoost) algorithm with a focus on hyperparameter tuning in a limited computing environment to find optimal hyperparameter values for the model. This study uses basic Google Colab with 12 GB RAM with a total dataset of 431,371 entries. The results obtained from this study conclude that hyperparameter tuning achieves optimal performance at n_estimators = 150 and max_depth = 15, resulting in 99.97% accuracy, 99.99% precision, 99.97% recall, and 99.98% F1 score, which proves the effectiveness of the model in detecting flooding attacks. The novelty of this study lies in the application and evaluation of hyperparameter tuning on the XGBoost algorithm in a resource-constrained environment to improve attack detection in SDN-VANET.

Corresponding Author:
Chairunisa Rahma Putri,
Department of Telecommunication System, Universitas Pendidikan Indonesia, West Java, Indonesia,
Email: *chairunisarahma@upi.edu

1. INTRODUCTION

According to the World Health Organization (WHO) in its Global Status Report on Road Safety 2023, approximately 1.19 million lives are lost annually due to traffic accidents [1]. One of the efforts to mitigate or reduce the number of traffic accidents that is being developed is Vehicular Ad Hoc Network (VANET) technology. VANET has a dynamic nature and its topology often changes rapidly because it consists of moving nodes [2]. VANET is a wireless network that allows vehicles to communicate with each other and exchange information directly (Vehicle-to-Vehicle, V2V) or communicate with surrounding infrastructure or Road-Side Units (Vehicle-to-Infrastructure, V2I) [3]. With technologies such as IEEE 802.11p or 5G NR-V2X, VANET is able to send automatic messages such as collision warnings, traffic jam information, and emergency stops [4]. Now VANET is increasingly developing with the integration of Software Defined Network (SDN) which allows infrastructure control by default because traditional VANET has limited scalability and security [5]. SDN is an extension of conventional networks that allows centralized control by network administrators [6]. SDN is divided into three layers, namely the data layer, the controller layer, and the infrastructure layer. The data layer is responsible for forwarding packets based on instructions from the controller. The control layer is responsible for routing and managing traffic. Meanwhile, the infrastructure layer provides hardware such as routers, switches, and firewalls or virtual networks that serve as a container for data layer operations in forwarding traffic based on instructions from the control layer [7]. With this separation of functions, SDN allows centralized management, and is easy to set up and monitor even from a distance. The integration of VANET with SDN gives birth to SDN-based VANET which enhances the advantages of VANET itself, such as scalability and security that increase and improve adaptability [8]. Figure 1 below is an illustration of an SDN-based VANET infrastructure.

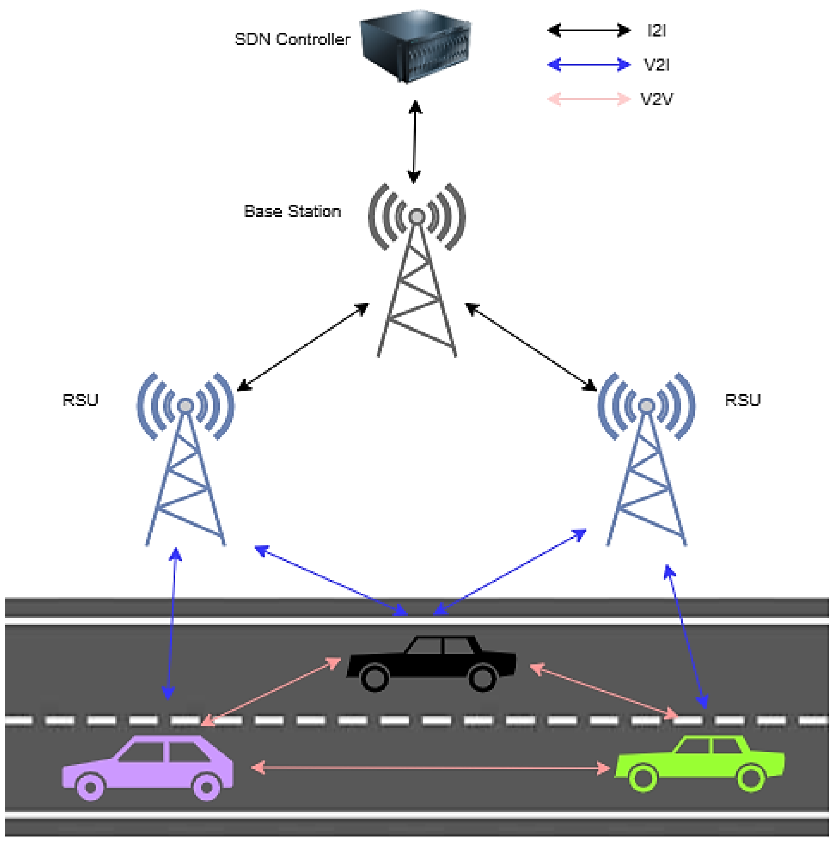


Figure 1. SDN-based VANET architecture consists of three main layers, namely data plane (vehicle), control plane (SDN controller), and infrastructure plane (base station and RSU)

In addition to improving scalability, security, and adaptability, SDN-based VANETs can also implement large-scale vehicular traffic infrastructure [9]. However, SDN-based VANETs face challenges in terms of network security, due to their centralized nature [10]. Network security attacks that have attracted much attention are flooding attacks, which include Denial of Service (DoS) or Distributed-Denial of Service (DDoS). Attacks such as flooding attacks can paralyze the control plane or Road-Side Unit by flooding it with excessive traffic, thereby disrupting service availability [11]. Control plane paralysis in SDN-based VANETs will disrupt vehicle traffic [12]. In addition, the disruption of services due to flooding attacks causes the transmission of important messages such as collision warnings or road condition information between vehicles to stop, which increases the risk of traffic accidents [13]. To overcome this problem, several researchers have proposed approaches, such as threshold-based approaches, which are non-machine learning approaches that compare certain metrics with a predetermined threshold [14]. However, this research discusses more about the Machine Learning (ML) approach in detecting flooding attacks on SDN-based VANET infrastructure networks.

Currently, Machine Learning (ML) based approaches are available that are able to analyze network traffic patterns, detect anomalies, and classify attacks more accurately [15]. Several studies have described several effective algorithms in detecting flooding attacks, one of which is Extreme Gradient Boosting (XGBoost). XGBoost is a machine learning algorithm that combines several decision trees into one powerful model, and each tree helps correct errors from the previous tree [16]. One study explains that the XGBoost algorithm is able to detect DDoS attacks very well on VANET infrastructure with the same accuracy, precision, recall, and F1-Score values, namely 97% [17]. Research [18] explains that XGBoost is able to provide an accuracy of 99.93%, precision of 99.55%, recall of 99.86%, in handling DoS attacks on SDN-based VANET infrastructure. In addition, research [19] also concludes that XGBoost is able to provide an accuracy of 99.84% in detecting DDoS attacks on SDN networks. However, most previous studies have focused on the performance of the XGBoost model without considering the efficiency of computational resources and the hyperparameter tuning process to find optimal values for the detection model. Furthermore, few studies have examined how resource limitations can affect the performance of flood attack detection models. In this study, the resource limitation referred to is the use of the free Google Colab with 12 GB of RAM and a limited runtime. This approach is important because much academic research and network security prototype testing is conducted using limited resources. Therefore, this study not only assesses the effectiveness of the algorithm but also examines the extent to which the algorithm remains consistent, efficient, and reliable under limited and non-ideal resource conditions.

This study aims to bridge this research gap by tuning hyperparameters on the XGBoost algorithm to detect flooding attacks in SDN-based VANETs. The main contributions of this study include exploring the impact of resource limitations on model performance in detecting attacks, evaluating the effect of hyperparameter combinations on performance metrics, and providing recommendations for optimal hyperparameter configurations on Google Colab for detecting flooding attacks. Specifically, this study adjusts two hyperparameters, namely n_estimators, and max_depth, to evaluate model performance through precision accuracy, recall, and F1-Score metrics in a resource-limited Google Colab environment. Thus, this study is expected to increase contributions in the development of more efficient and adaptive flooding attack detection and prevention systems for SDN-based VANET networks.

2. RESEARCH METHODS

This study employs an experimental approach using ML through the XGBoost algorithm to detect flooding attacks in SDN-based VANET networks. The experiment was conducted in the free version of Google Colab, which provides 12 GB of RAM and a limited runtime session of approximately 12 hours. In addition, the hardware used is a 10th generation Intel i3 1.2 GHz processor. The implementation was developed using Python 3.12.11, scikit-learn 1.6.1, numpy 2.0.2, pandas 2.2.2, matplotlib 3.10.0, and XGBoost 3.0.5.

The dataset used in this study represents SDN-based VANET network traffic, containing recordings of flooding attacks and normal traffic. Each packet has characteristics such as the total number of bytes transferred, TCP signature, packet interarrival time, and other related traffic characteristics. This dataset was obtained from packet capture using tshark version 3.2.3 and preprocessed into CSV format, consisting of 9 features and 431,371 total samples.

Flooding attack detection in this study is formulated as a binary classification problem, with label 0 representing normal traffic and 1 representing traffic with flooding attacks. The random seed value is set at 42 to ensure reproducibility of the results. To better understand the research flow, Figure 2 below displays the research flow presented in the form of a flowchart.

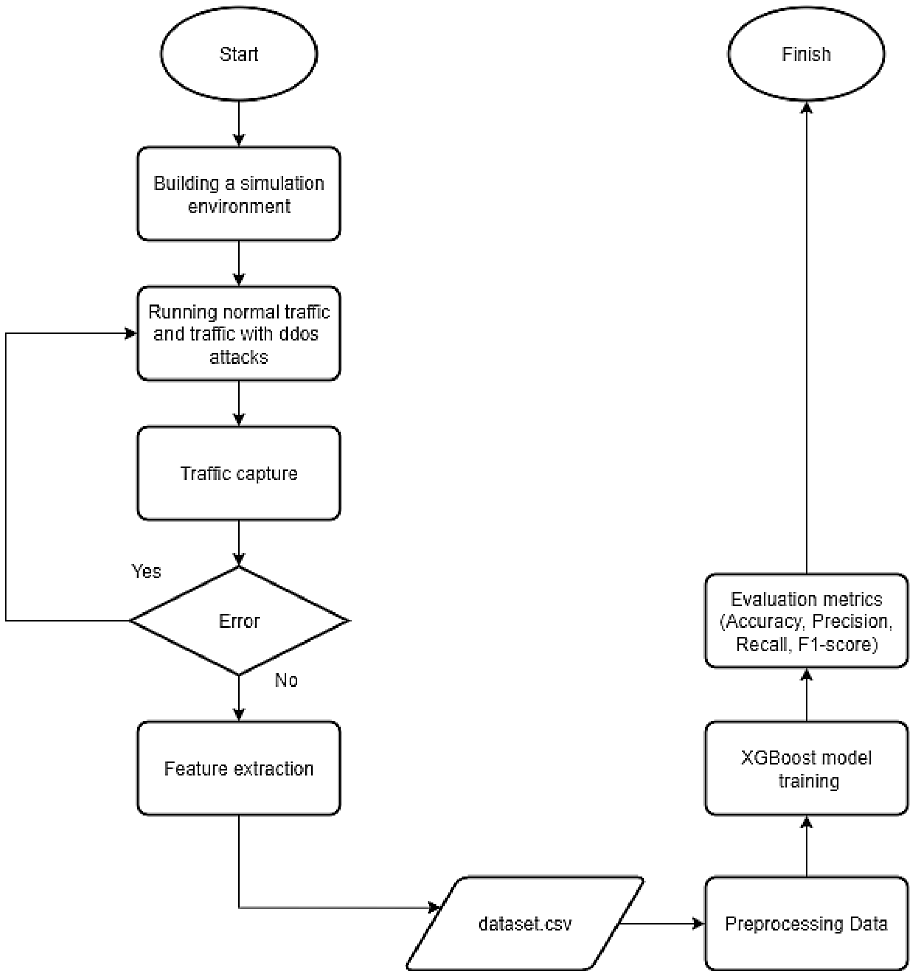
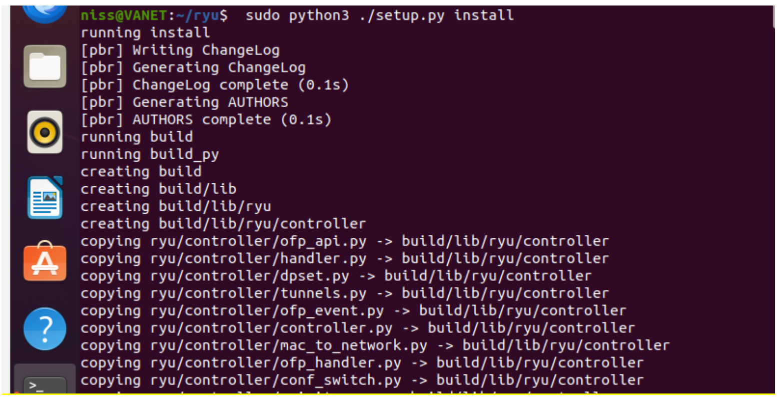

Start
Building a simulation environment
Running normal traffic and traffic with ddos attacks
Traffic capture
Yes
Error
No
Feature extraction
dataset.csv
Preprocessing Data
XGBoost model training
Evaluation metrics (Accuracy, Precision, Recall, F1-score)
Finish

Figure 2. Research flowchart. This diagram shows the process starting from SDN-VANET simulation, data preprocessing, feature selection, hyperparameter tuning, to model evaluation

2.1 Build SDN-Based VANET Environment

The initial step of this study is to create an SDN-based VANET simulation environment using VirtualBox version 7.2.2 as a hypervisor for Ubuntu 20.04.6. The number of cars in the SDN-based VANET simulation that was run was 4 cars with 1 rsu, and with car3 as the attacker. The creation of this simulation environment includes installing hping3 version 3.0.0-alpha-2, tcpdump version 4.9.3, net-tools version 1.60, mininet-wifi version 2.6, tshark version 3.2.3 and Ryu controller version 4.34, then running normal traffic or flood attack traffic simultaneously with the ryu controller, and after that, capturing packets using tcpdump. Figure 3 below shows the commands for installing the Ryu controller. The installation steps in Figure 3 were executed after cloning from the public Ryu repository.

Normal traffic is run by sending normal packets via ping for icmp packets or iperf for udp from car1 to car2, while sending traffic with a flood attack is sent from car3 to rsu using the hping3 tool. The command used when using hping3 to send traffic with a flood attack is “car3 hping3 –flood -S -p 80 rsu1”. Capture packets simultaneously with normal traffic and flooded traffic using tcpdump. Tcpdump will capture packets and save the data in PCAP format. The command to capture packets using tcpdump is "sudo tcpdump -i rsu1-wlan1 -s 0 -w /tmp/traffic.pcap".



```
niss@VANET:~/ryu$  sudo python3 ./setup.py install
running install
[pbr] Writing ChangeLog
[pbr] Generating ChangeLog
[pbr] ChangeLog complete (0.1s)
[pbr] Generating AUTHORS
[pbr] AUTHORS complete (0.1s)
running build
running build_py
creating build
creating build/lib
creating build/lib/ryu
creating build/lib/ryu/controller
copying ryu/controller/ofp_api.py -> build/lib/ryu/controller
copying ryu/controller/handler.py -> build/lib/ryu/controller
copying ryu/controller/dpset.py -> build/lib/ryu/controller
copying ryu/controller/tunnels.py -> build/lib/ryu/controller
copying ryu/controller/ofp_event.py -> build/lib/ryu/controller
copying ryu/controller/controller.py -> build/lib/ryu/controller
copying ryu/controller/mac_to_network.py -> build/lib/ryu/controller
copying ryu/controller/ofp_handler.py -> build/lib/ryu/controller
copying ryu/controller/conf_switch.py -> build/lib/ryu/controller
```

Figure 3. Installation process of the Ryu SDN controller on Ubuntu 20.04 using Python 3.7

After successfully capturing packets in PCAP format, the next step is to extract the PCAP file into CSV format using tshark, which allows each traffic flow to be represented numerically. The command used to convert PCAP format to CSV using tshark is “tshark -r /tmp/traffic.pcap -T fields -E header=y -E separator=, \ -e frame.time_epoch -e ip.src -e ip.dst -e ws.col.Protocol \ -e tcp.srcport -e tcp.dstport -e udp.srcport -e udp.dstport -e frame.len > /tmp/dataset.csv”. The resulting dataset.csv will contain 9 extracted features, such as frame.time_epoch, ip.src, ip.dst, _ws.col.Protocol, tcp.srcport, tcp.dstport, udp.srcport, udp.dstport, and frame.len. This CSV data will then be used as input for the preprocessing stage before being applied to the machine learning model.

2.2 Data Preprocessing

Figure 4 above shows a flowchart of the most crucial step, namely data preprocessing. The SDN-based VANET traffic dataset extracted into CSV format undergoes preprocessing. This process is carried out using the free Google Colab tool, starting from data cleaning to oversampling. This preprocessing stage includes the data cleaning process, or ensuring whether the dataset contains missing values. In this study, no missing values were found in the dataset. If the dataset contains missing values, they will be removed using df = df.dropna(). To ensure whether the dataset contains missing values or not, this study uses print(df.isnull().sum().sum()) to confirm whether or not there are missing values. Because the algorithm used is XGBoost, which is based on decision trees, normalization or feature scaling methods are not used, considering that the algorithm is not sensitive to scale differences between features. The next step in the preprocessing stage is label encoding, which marks the data using a binary classification, with 0 representing normal traffic and 1 representing traffic with a flooding attack.

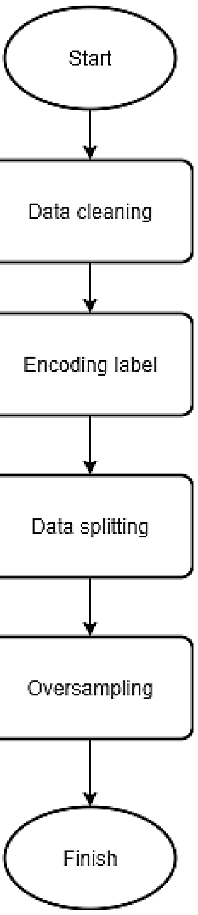
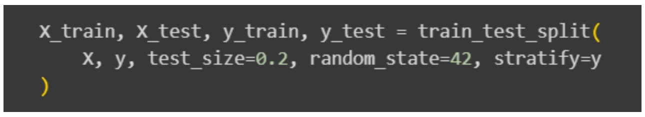


Figure 4. Data preprocessing flowchart showing the steps of cleaning, label encoding, data splitting, and oversampling before model training

After completing the label encoding stage using binary classification, the next step is to perform data splitting to separate the training and test sets. The goal is to train the model on the training data and then test it on the test data. The training to test data ratio is 80:20, and a random seed of 42 is used to ensure reproducibility of the results. This data split is performed using a train-test split. The train-test split divides the data into test and training data and produces four outputs: x_test, y_test, x_train, and y_train. The train-test split function was chosen because it is easier and faster to implement compared to K-fold. The random seed value chosen is a common value used in the data science community. Figure 5 below shows the source code used in the data split step.



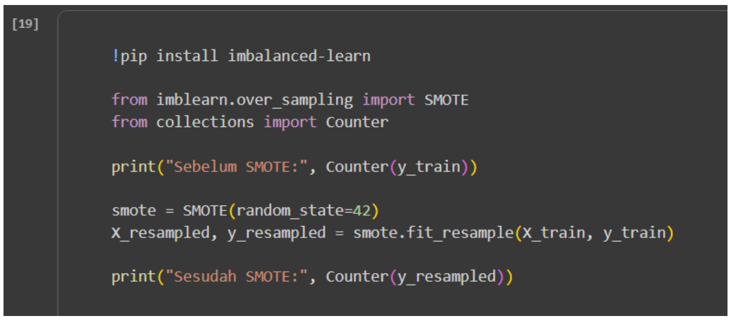
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Figure 5. Python code for splitting the dataset into training (80%) and testing (20%) sets using the train_test_split function

Before proceeding to the model training and testing phase, oversampling was performed. Based on Table 1, the normal traffic data and the flood attack traffic were unbalanced. Therefore, oversampling using the Synthetic Minority Oversampling Technique (SMOTE) was applied using the balanced-learn library version 0.14.0, which aims to increase the volume of minority data. The amount of training data before the oversampling process was 266.831 for attack traffic data, and 78,265 for normal traffic. After the oversampling process, the number of attack traffic data and normal traffic data was balanced at 266,831. This oversampling was performed on the training data so that the model could be trained with balanced data and prevent the model from being biased towards minority data. Figure 6 shows the steps for oversampling using SMOTE on the training data.

Table 1. Classified dataset. The dataset is categorized into normal and attack traffic.

Normal Traffic	Traffic with Flooding Attacks	Total Data
97.831	333,540	431,371



```
[19]
!pip install imbalanced-learn

from imblearn.over_sampling import SMOTE
from collections import Counter

print("Sebelum SMOTE:", Counter(y_train))

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

print("Sesudah SMOTE:", Counter(y_resampled))
```

Figure 5. Applying SMOTE to balance the training dataset by generating synthetic samples for the minority class. generating synthetic samples for the minority class

2.3 Implementation of the XGBoost Algorithm with a Limited Resource Scheme

After the dataset has passed the preprocessing stage, the next step is to implement the XGBoost algorithm to detect flooding attacks on SDN-based VANETs using the available dataset. XGBoost is a decision tree-based ensemble learning algorithm. XGBoost works by building several trees in stages, where each tree tries to minimize the error (loss) of the previous tree [20]. The implementation was carried out in the same environment as previously described (Google Colab Free, 12 GB RAM, Python 3.12.11, scikit-learn 1.6.1, XGBoost 3.0.5), and using the same random seed, namely 42. Mathematically, this process can be described by the minimized loss function as Equation (1).

$$L^{(t)} = \sum_{i=1}^{n} l\left(yi, yi^{(t-1)} + ft(xi)\right) + \Omega\,(ft) \tag{1}$$

with: l : loss function
yi : original function
$yi^{(t-1)}$: model predictions on previous iterations
$ft(xi)$: new decision tree added at iteration t
$\Omega\,(ft)$: regularization that controls tree complexity to prevent overfitting

Several hyperparameters were adjusted, including the number of trees (n_estimators) and tree depth (max_depth), as part of the hyperparameter tuning. These adjustments took into account the testing environment's resources, a standard Google Colab with 12 GB of memory, intended to measure model consistency. The following details the testing environment and the parameters used. Table 2 below shows details of the resources used and what hyperparameters will be adjusted to find the optimal values in the model.

Table 2. Configuration of compute resource and hyperparameters used in the experiment

Compute Resource	RAM (GB)	N_estimators	Max_depth
Google Colab Basic/Free	12	50	5
		100	10
		150	15
		200	20

The evaluation metrics examined include accuracy, which measures the proportion of correct predictions; precision, which measures the model's ability to detect attacks without excessive false positives; recall, which measures the model's ability to detect all attacks with minimal false negatives; and finally, the F1 score, which measures the balance between precision and recall [21]. Equations (2) - (5) are the mathematical for these metrics.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

$$Precision = \frac{TP}{TP+FP} \tag{3}$$

$$Recall = \frac{TP}{TP+FN} \tag{4}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + recall} \tag{5}$$

with: TP : true positive cases detected positive
TN : true negative cases detected negative
FP : negative cases falsely detected as positive
FN : positive cases falsely detected as negative

For more detailed explanation, the four metrics mentioned above, particularly in the context of flooding attack detection, comprehensively illustrate the performance of an attack detection model, not just from a single perspective. Accuracy is used to determine the extent to which the model is able to provide correct predictions overall, both in detecting normal traffic and flooding attack traffic. However, if the dataset used is not balanced between normal traffic and traffic with flooding attacks, then the accuracy metric alone is not sufficient to measure the model's performance. Therefore, other metrics such as precision and recall are needed to complement each other. Precision measures the level of accuracy of the model in identifying attacks, namely showing how accurately the model detects attacks that are actually attacks. Precision is very important because it minimizes detection errors, as these errors can impact network operations. Meanwhile, recall measures the model's ability to detect attacks that actually occur. A high recall value indicates that the model is sensitive to attacks, so that not many attacks go undetected. Finally, the F1-Score serves to assess the balance between recall and precision. The F1-Score metric is very relevant to be applied to models that aim to detect an attack, such as flooding attacks, because an ideal model should detect as many attacks as possible without frequently making detection errors. Thus, the F1-Score provides a fairer assessment of overall model performance than relying solely on a single model. Overall, observing these four metrics helps ensure that the trained model is not only statistically accurate but also effective and reliable when applied to detection systems, particularly in SDN-based VANET networks.

3. RESULTS AND DISCUSSION

The results of accuracy, precision, recall, and F1-score presented in Table 3 below are the results of hyperparameter tuning on two parameters, namely n_estimators and max_depth, which aim to find the optimal parameters and metrics (accuracy, precision, recall, and F1_Score) for the XGBoost model that has been trained based on the available dataset.

Table 3. Metrics result

N_estimators / max_depth	Accuracy	Precision	Recall	F1-Score	Processing Time
50 / 5	99.79%	99.98%	99.75%	99.86%	10.00s
100 / 10	99.96%	99.98%	99.97%	99.97%	13.35s
150 / 15	99.97%	99.99%	99.97%	99.98%	20.67s
200 / 20	99.96%	99.98%	99.97%	99.98%	23.50s

Table 3 above shows the results of hyperparameter tuning experiments with four different variations of n_estimators and max_depth. In the table, the accuracy, precision, recall, and F1-score metrics are all above 99.79, indicating the model can effectively detect flooding attacks, outperforming earlier VANET-based XGBoost studies that reported average performance around 97% [17]. As the number of trees increases, processing time also increases, with processing time increasing from 10 for the 50/5 configuration to 23.50 for the 200/20 configuration. This indicates that the higher the parameter values, the more complex the resulting model. The model's hyperparameters were set to assess the model's performance. These hyperparameters were n_estimators and max_depth. There were four variations of n_estimators and max_depth. The model was evaluated using four metrics: accuracy, precision, recall, and F1-score, each yielding varying results.

Based on the test results, there was a trend of increasing performance across accuracy, precision, recall, and F1-score when n_estimator was increased from 50 to 150. However, after the n_estimator value reached 150, model performance began to decline across all evaluation metrics. This indicates that increasing the hyperparameter value does not always improve model quality but can lead to overfitting and resource inefficiency, an aspect that was not explicitly analyzed in previous SDN-based VANET studies employing XGBoost [18]. Furthermore, the model's performance increased with increasing training time. It was noted that at n_estimators = 50 and max_depth = 5, the model training time was only around 10 seconds, and increased to 20.67 seconds at n_estimators = 150 and max_depth = 15, or about 110% of the starting point. Thus, although increasing the hyperparameter values and can improve metrics, the computational cost also increases. Therefore, the choice of values for hyperparameters is very important for the model to have good performance without sacrificing efficiency. In the final configuration, namely n_estimators = 200 and max_depth = 20, the model performance began to decline slowly but was still accompanied by an increase in training time, indicating increasing model complexity, and increasing the occurrence of overfitting and inefficiency in resources. Thus, the configuration of n_estimators = 150 and max_depth = 15 can be recommended as the optimal setting because after that the model began to gradually experience a decrease in performance but the training time continued to increase.

Previous studies, such as the study conducted by Bappa Muktar et al. [17], reported that XGBoost achieved the same accuracy, precision, recall, and F1-Score values, namely 97% when a DDoS attack occurred on a VANET network. In that study, the dataset used was the result of a simulation using tools such as the Simulation of Urban Mobility (SUMO) to provide a realistic picture of the VANET infrastructure. However, that study did not apply the SDN paradigm in the VANET infrastructure, and did not consider hyperparameter tuning, let alone using limited resources when training the model, which is the main focus of this study.

Another study by Nadhir Fachrul Rozam et al. [19], concluded that XGBoost has a higher accuracy than other algorithms studied, namely Support Vector Machine (SVM), and Random Forest, which is 99.84%, in detecting DDoS attacks on SDN networks. The study used 4 different datasets, and used ONOS as the SDN controller. However, the study did not explain the integration of VANET with SDN and the application of hyperparameter tuning for model optimization. In addition, the study also did not discuss resource usage when training the model.

Meanwhile, research from Adi El-Dalahmeh, et al. [18], explains that XGBoost provides an accuracy of 99.93%, 99.55% for precision, 99.86% for recall, in handling DoS attacks on SDN-based VANET networks with the number of data in the dataset for normal traffic of 3,078,250, and for traffic with DoS attacks of 587,521. The study concluded that XGBoost is superior to other methods, such as K-Nearest Neighbor (KNN), and LSTM-AE. However, the study has not practiced hyperparameter tuning to the XGBoost model, and it is not explained whether the model was trained in a limited environment or not.

Based on the comparison of the three studies, this research contributes to filling the research gap by applying hyperparameter tuning to XGBoost in an SDN-based VANET environment using limited resources (Google Colab free). This approach not only optimizes model performance but also demonstrates training efficiency in limited resources, which has not been widely explored by other researchers. However, this study certainly has limitations, including the dataset used is still relatively small when compared to other studies, so the generalization of the results needs to be tested on a larger dataset. In addition, this study only explains about flooding attacks, so testing other types of attacks such as spoofing or blackhole attacks can be developed in future research. Furthermore, since the dataset used in this study is the result of an SDN-based VANET simulation, it is necessary to validate the model using real-world network captures to assess its generalization.

4. CONCLUSION

This study demonstrates that XGBoost is a robust model for detecting flooding attacks such as DoS or DDoS in SDN-based VANET environments, consistent with findings in previous studies by Bappa Muktar et al., Nadhir Fachrul Rozam et al., and Adi El-Dalahmeh et al.. However, unlike these studies, this study demonstrates that hyperparameter optimization or tuning in resource-constrained environments can further improve model efficiency and detection performance. The proposed configuration (n_estimators = 150, max_depth = 15) achieves 99.97% accuracy, 99.99% precision, 99.97% recall, and 99.98% F1-score. These results validate the research objective of improving model performance through hyperparameter tuning despite computational limitations. The novelty of this study lies in the demonstration of effective hyperparameter tuning for XGBoost in resource-constrained environments, expanding the practical application of ML-based network security in VANET systems.

REFERENCE

[1] World Health Organization, *Global Status Report on Road Safety 2023*, Geneva, Switzerland: WHO, 2023.
[2] M. Priya, S. Pravin Kumar, S. Rajalakshmi, R. Sangeetha, and S. Anuradha, "Vehicle Ad-Hoc Network for Road Traffic and Accident Preventing," *International Journal for Research & Development in Technology*, vol. 7, no. 4, Apr. 2017.
[3] A. Dutta, A. Nandi, B. Das, and A. Chakraborty, "A Comprehensive Review of Recent Developments in VANET for Traffic, Safety & Remote Monitoring Applications," *Journal of Network and Systems Management*, vol. 32, no. 4, p. 73, Oct. 2024. https://doi.org/10.1007/s10922-024-09853-5.
[4] R. S. Sandesh and K. Santhosh, "Revolutionizing vehicle communication: A comprehensive exploration of technologies for enhanced road safety and autonomous vehicles," *Alexandria Engineering Journal*, vol. 129, pp. 976–997, Oct. 2025. https://doi.org/10.1016/j.aej.2025.08.008.
[5] W. B. Jaballah, M. Conti, C. Lal, and D. Djenouri, "A Survey on Software-Defined VANETs: Benefits, Challenges, and Future Directions," *arXiv*, May 2019. https://doi.org/10.48550/arXiv.1904.04577.
[6] S. Thangavel, K. C. Sunkara, and S. Srinivasan, "Software-Defined Networking (SDN) in Cloud Data Centers: Optimizing Traffic Management for Hyper-Scale Infrastructure," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 3, no. 1, pp. 29–42, 2022.
[7] R. Wazirali, R. Ahmad, and S. Alhiyari, "SDN-OpenFlow Topology Discovery: An Overview of Performance Issues," *Applied Sciences*, vol. 11, no. 15, p. 6999, Jul. 2021. https://doi.org/10.3390/app11156999.
[8] M. A. Setitra, "Detection of DDoS attacks in SDN-based VANET using optimized TabNet," *Computer Standards & Interfaces*, vol. 90, 2024. https://doi.org/10.1016/j.csi.2024.103845.

[9] M. Arif, G. Wang, O. Geman, V. E. Balas, P. Tao, A. Brezulianu, and J. Chen, "SDN-based VANETs, Security Attacks, Applications, and Challenges," *Applied Sciences*, vol. 10, no. 9, p. 3217, May 2020. https://doi.org/10.3390/app10093217.

[10] R. Sultana, S. M. M. Rahman, and A. Anwar, "Security of SDN-based vehicular ad hoc networks: State-of-the-art and challenges," *Vehicular Communications*, vol. 27, p. 100284, Jan. 2021. https://doi.org/10.1016/j.vehcom.2020.100284.

[11] H. Karthikeyan and G. Usha, "Real-time DDoS flooding attack detection in intelligent transportation systems", *Computer and Electrical Engineering*, vol. 101, p. 107995, Jul. 2022. https://doi.org/10.1016/j.compeleceng.2022.107995.

[12] V. Karthik, R. Lakshmi, S. Abraham, and M. Ramkumar, "Residual-based temporal attention convolutional neural network for detection of distributed denial of service attacks in software-defined network integrated vehicular adhoc network," *International Journal of Network Management*, vol. 34, no. 3, p. e2256, May 2024. https://doi.org/10.1002/nem.2256.

[13] Z. El-Rewini, K. Sadatsharan, D.F. Selvaraj, S.J. Plathottam, and P. Ranganathan, "Cybersecurity challenges in vehicular communications", *Vehicular Communications*, Vol. 23, p. 100214, Jun. 2020. https://doi.org/10.1016/j.vehcom.2019.100214.

[14] D. Rani and M. K. Soni, "Efficient Detection of DDoS Attack Using Threshold Based Technique in VANETs," *SSRN Electronic Journal*, 2024. https://doi.org/10.2139/ssrn.4485752.

[15] M. H. Thwaini, "Anomaly Detection in Network Traffic using Machine Learning for Early Threat Detection," *Data and Metadata*, vol. 1, p. 72, Dec. 2022. https://doi.org/10.56294/dm202272.

[16] M. Imani, S. Poria, and H. Rezaei, "Comprehensive Analysis of Random Forest and XGBoost Performance with SMOTE, ADASYN, and GNUS Under Varying Imbalance Levels," *Technologies*, vol. 13, no. 3, p. 88, Feb. 2025. https://doi.org/10.3390/technologies13030088.

[17] B. Muktar, V. Fono, and A. Nouboukpo, "Machine Learning-Based Detection of DDoS Attacks in VANETs for Emergency Vehicle Communication," *Computers, Materials & Continua*, vol. 85, no. 3, pp. 4705–4727, 2025. https://doi.org/10.32604/cmc.2025.067733.

[18] A. El-Dalahmeh, A. Abu-Shareha, and M. Alzoubi, "An Intrusion Detection System Using the XGBoost Algorithm for SDVN," *Advances in Computational Intelligence Systems*, vol. 1453, pp. 390–402, Feb. 2024. https://doi.org/10.1007/978-3-031-47508-5_31.

[19] N. F. Rozam and M. Riasetiawan, "XGBoost Classifier for DDoS Attack Detection in Software Defined Network Using sFlow Protocol," *International Journal of Advanced Science, Engineering and Information Technology*, vol. 13, no. 2, pp. 718–725, Apr. 2023. https://doi.org/10.18517/ijaseit.13.2.17810.

[20] Z. A. Ali, R. Raj, and M. N. Mohammed, "eXtreme Gradient Boosting Algorithm with Machine Learning: a Review," *Academic Journal of Nawroz University*, vol. 12, no. 2, pp. 320–334, May 2023. https://doi.org/10.25007/ajnu.v12n2a1612.

[21] N. Abedzadeh and M. Jacobs, "A Reinforcement Learning Framework with Oversampling and Undersampling Algorithms for Intrusion Detection System," *Applied Sciences*, vol. 13, no. 20, p. 11275, Oct. 2023. https://doi.org/10.3390/app132011275.